ACSE 6 Report

Introduction

Class

The purpose of this project is to use MPI to solve the wave equation in parallel. Because one-dimensional arrays or two-dimensional arrays cannot express boundary conditions easily and clearly, a Matrix class is created, which including information like values, rows, columns, and boundary conditions. In addition, because this project uses both stripe decomposition and grid decomposition, and these two decompositions have their own unique functions, two subclasses of Matrix called strip and Grid also be created, whose parameters are same to Matrix.

Initial conditions

In order to generate waves, it is necessary to disturb the initial state. The disturbance strategy of this project is the same as the disturbance strategy provided by the Serial_Wave_Equation.cpp.

Boundary conditions

The matrix can take one of three boundary conditions including Dirichlet boundary, Neumann boundary and periodic boundary. For Dirichlet boundary, the value of the matrix on boundaries are 0. As for Neumann boundary, It satisfies the formula $\nabla u \cdot n = 0$, and n is a unit vector normal to the boundary. The periodic boundary means the one side of the matrix has communication with another side of the matrix. For example, the top boundary will communicate with the bottom boundary and the left boundary will communicate with the right boundary.

Decomposition method

The much easier method is stripe decomposition. In this project, we will divide the rows in p (the number of the processes) sections. Each process will process part of the total data simultaneously to improve the calculation speed.

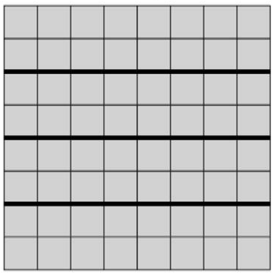


Figure 1: stripe decomposition

The other decomposition method is grid decomposition. In fact, it is the method that uses

stripe decomposition in both rows and columns directions. Therefore, before doing this decomposition, firstly we should divide your p processes into a grid of size m x n and try to make sure that m and n are integers as close to each other as possible. Then use stripe decomposition to divide the rows into m sections and divide the columns into n sections.

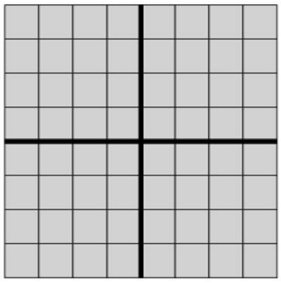


Figure 2: grid decomposition

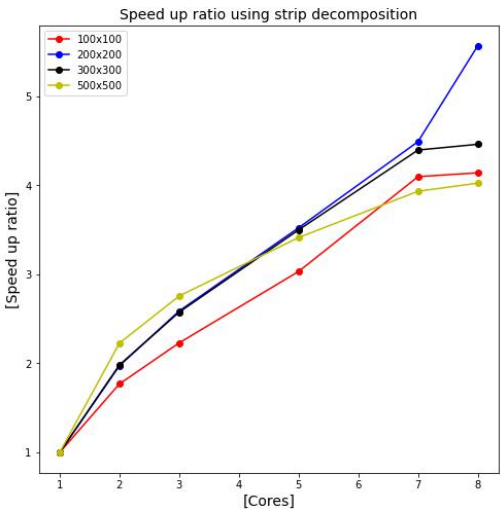
Post processing

In this project, each process writes its own .dat file to the folder named out. Because we want to obtain the final results from all processes, the functions collect all processes sub-results and combine them to a big array have been provided in Python file. The user can adjust some parameters like p, imax, jmax to get the summarized result which is still .dat type file. Then, according to the summarized result, the functions that convert array to picture and combine pictures to a video also be provided in Python file. After running them, the user can get a video and many pictures to see the wave changes more intuitively.

Analysis

In this example, because when we change the imax or jmax, the number of the iterations will change as well, comparing the time cost from different sizes of the array will be inappropriate. In analysis, we only compare the speed-up ratio and parallel efficiency.

Speed up ratio



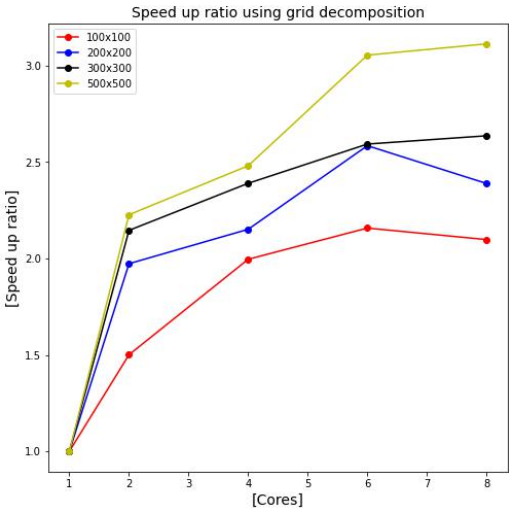


Figure 3: Speed up ratio using stripe and grid decomposition

From figure 3, we can find that the speed-up ratio always increases with the number of cores. However, the rate of growth gradually decreases. Based on Amdahl's law, law, we know the speed-up ratio can be expressed as

$$S = \frac{1}{1 - f + \frac{f}{N}}$$

which f is the fraction of the code that executing in parallel. The reason is that when the number of cores reaches a certain number, the serial part of the code occupies most of the running time. Simply increasing the number of cores cannot reduce the running time of the serial part of the code, so the growth rate decreases. In addition, we can find that for the same number of cores and size of arrays, the speed-up ratio of stripe decomposition is bigger than the grid decomposition. The reason is that for a small number of cores and small size arrays, the communication cost of stripe decomposition is smaller than grid decomposition's. When the number of cores rapidly increases, the grid decomposition will be more efficient.

Parallel efficiency

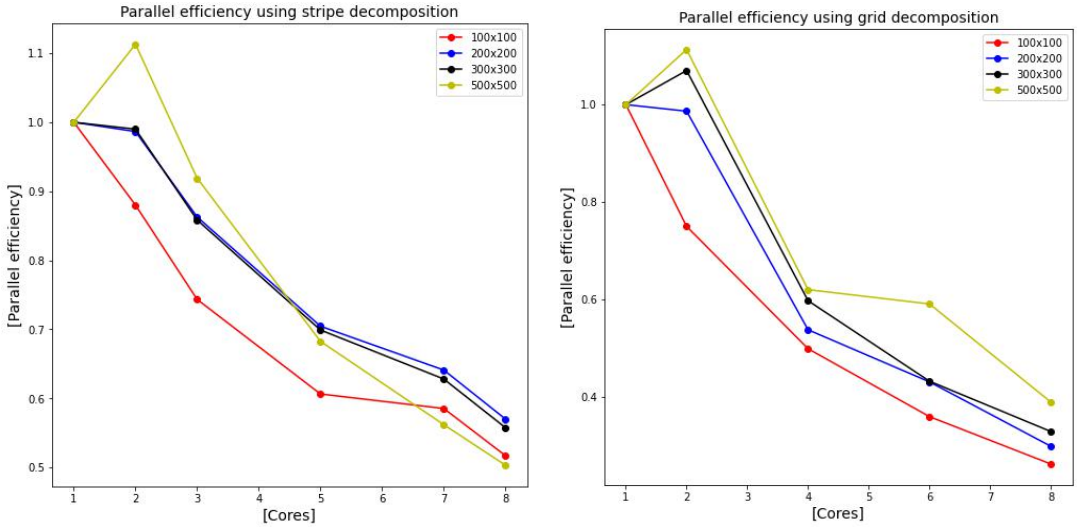


Figure 4: Parallel efficiency using stripe and grid decomposition

From figure 4, we can find that parallel efficiency drop as the number of cores used increases. In addition, from the theory, the bigger problems solving using the same number of cores usually have higher efficiency. Most of the curves in the figure coincide with this theory, however, some curves don't coincide with this theory. I think the reason is the errors in measuring time. We can reduce the error by measuring time multiple times.